6809 FLEX™ Utilities

INTRODUCTION

The utility programs documented herein are written to be position independent. Each program will run correctly, without modification, from any location in user memory. As a default, each program runs in the FLEX™ 9.0 Utility Command Space at location $C100. The RUN utility may be used to load and execute the programs at other memory locations, if desired.

This manual is designed to be taken apart and the pages added to the "User's Manual" section of the "6809 FLEX™ Operating System" manual.

In addition to the command files, the source for these utilities is supplied on the diskette. Users may customize these programs for their own end use. However, the legal protection of the rights of Technical Systems Consultants, Inc. over the software extends, by law, to include such modified versions.

CHECK

The CHECK utility is used to compare two disk files. The result of the comparison will be reported to the terminal.

DESCRIPTION

The general syntax of the CHECK command is:

CHECK,<file spec 1>,<file spec 2>

where the file specs default to a TXT extension and to the working drive. File one will be read and compared against file two one character at a time. The files may be text or binary files. The result of the comparison will be reported to the terminal (files are identical or not). An example follows:

+++CHECK,REPORT1,REPORT2

This command line would cause the file named REPORT1.TXT on the working drive to be compared to the file named REPORT2.TXT.

CMPMEM

The CMPMEM command compares the contents of a binary file on the disk to the contents of memory where it should be loaded. This is useful for program debugging and memory problem detection.

DESCRIPTION

The general syntax of the CMPMEM command is:

```
CMPMEM,<file spec>
```

where the file spec defaults to a BIN extension and to the working drive. The file specified will be read just as if it were to be loaded into memory, but instead, each byte will be compared to what already exists in memory. If any differences are found, they will be printed out as the address, followed by the data in memory at that location, followed by the data from the disk file. All differences will be printed on the output device. An example follows:

```
+++CMPMEM,FENCE
```

This would cause the file named FENCE.BIN on the working drive to be read and compared to the actual memory contents throughout the load address range of the file.

CONTIN

The CONTIN command is intended for use in repeating or complex EXEC command files. It prompts the terminal for a YES or NO response for continuing that file's execution.

DESCRIPTION

CONTIN

Executing CONTIN will cause the message "CONTINUE (Y-N)? " to be displayed on the terminal. A "Y" response will cause the EXEC program to execute the next command in the command file. An "N" response will cause FLEX to regain control and the EXEC program will be halted. This utility is useful for incorporating into EXEC command files which repeat themselves (by calling itself as the last line of the command file). The CONTIN command provides a mechanism for escape from this ever repeating type of command file.

DIR

The DIR utility is similar to the CAT command but displays all directory information associated with the file. This command gives a detailed look at the disk directory.

DESCRIPTION

The general syntax of the DIR command is:

DIR[,<drive list>][,<match list>]

where <drive list> and <match list> are the same as described in the CAT command. Each file name is listed with its file number, starting disk address in hex (track-sector), ending disk address, and file size in number of sectors. In addition, the file creation date and attributes are also displayed. Following the file name is an indication as to whether or not the file is a random file. At the end of the DIR list, a disk file use summary is printed, giving the total number of files, the number of sectors used by those files, the remaining number of sectors (free sectors), and the size of the largest file found on the disk. The "file number" associated with a file represents that file's location in the directory, so the file numbers may not be consecutive if a lot of files have been deleted from the disk or a match list was specified. A few examples follow:

```
+++DIR
+++DIR,1,A.T,FR
```

The first example would list all files on the working drive. The second example would list only those files on drive 1 whose names begin with "A" and extensions begin with "T", as well as those files whose names start with "FR".

DUMP

The DUMP utility is used for dumping the contents of a file, one sector at a time, in both hex and ASCII characters. It can be used as a disk debugging aid or to clarify the exact format of disk files.

DESCRIPTION

The general syntax of the DUMP command is:

```
DUMP,<file spec>
```

where <file spec> specifies the file to be dumped and defaults to a BIN extension. As each sector is displayed it will be preceded by two, 2 digit numbers, the first being the hex value of the track number, the second being the sector number of the sector being dumped. Each data line will contain 16 hex digits representing the data followed by the ASCII representations of the data. All non-printable characters are displayed as underscores (_). An example follows:

```
+++DUMP,FILE55
```

This would cause the contents of each one of the sectors contained in the file named FILE55.BIN to be dumped to the output device.

ECHO

The ECHO command is a very useful utility for incorporation into EXEC command files. It allows the echoing of ASCII strings to the terminal.

DESCRIPTION

The general syntax of the ECHO command is:

```
ECHO,<string>
```

where <string> is any string of printable characters terminated by a carriage return or end of line character. A few examples of the ECHO command follow:

```
+++ECHO,THE COPY PROCESS IS STARTING
+++ECHO,TERMINAL 12
```

The first example would print the string "THE COPY PROCESS IS STARTING" on the terminal. The second example would print "TERMINAL 12". It is often useful to use ECHO in long EXEC command files to send informative messages to the terminal to tell the operator of the status of the EXEC operation.

EXTRACT

The EXTRACT utility is used to create a file from other files or segments of files. It is not necessary to copy the segments to scratch files and concatenate them. A command file is used to tell EXTRACT which lines are to be read from the files and concatenated to form the new file. Raw text may also be copied from the command file to the file being created.

DESCRIPTION

The general syntax of the EXTRACT command is:

EXTRACT,<new file>,<command file>

where <new file> is the specification of the file being created, and <directive file> is the name of a text file containing the directives. The <new file> must not already exist. The default extension for each of these files is TXT. EXTRACT reads the directive file, processing the directives in the order that they appear in the file, and creates the new file in accordance with the directives.

DIRECTIVES

A directive is a line in the directive file which starts with a right parenthesis, ")", in column 1. A line in the directive file which does not contain a ")" in column 1 is considered raw text and is immediately copied to the file being created. A directive has the following general form:

)<file spec><optional line list>

The <file spec> is a FLEX file specification. If no extension is specified, TXT is assumed. The file must, of course, already exist. The <optional line list> indicates which lines from the file are to be extracted and copied to the file being created. If no <optional line list> is specified, the entire file is copied. The <optional line list> consists of a series of single line numbers or line number ranges separated by commas. A line number range is a pair of line numbers separated by a hyphen (starting line-ending line). If the "starting line" is not specified, the beginning of the file is assumed. If the "ending line" is not specified, the end of the file is assumed. The line numbers and ranges in the line list do not have to be in ascending order; the file will be rewound, if necessary, in order to reach the line(s) being copied. If the last character on a directive line is a comma, the directive is assumed to continue starting in column 1 of the next line. Thus, directives may be continued across multiple lines. An

example follows:

```
EXTRACT NEW,INPUT
```

This command tells EXTRACT to create the file "NEW.TXT" from parts of the files mentioned on directives contained on the directive file "INPUT.TXT".

Assume that the directive file for the above example contains:

```
)FILEONE,5,7-10,2-3,50-
ADDITIONAL TEXT TO BE INSERTED
)FILETWO
)FILEONE,-10,15,20,
30,40-80
```

The file NEW will then contain, in order, lines 5, 7 through 10, 2 through 3, and 50 through the end of the file from the file FILEONE.TXT; the line ADDITIONAL TEXT TO BE INSERTED; all of FILETWO.TXT; and lines from the beginning of the file through line 10, lines 15, 20, 30, and 40 through 80 from the file FILEONE.TXT.

FILES

The FILES utility is similar to the CAT command but displays only the file names and extensions. This command is useful for getting a short and quick report of the directory contents.

DESCRIPTION

The general syntax of the FILES command is:

```
FILES[,<drive list>][,<match list>]
```

where <drive list> and <match list> are the same as described in the CAT command. The file names will be listed across the page and in a columnar fashion. The number of names displayed per line is determined by the TTYSET Width parameter. If the Width is zero, 80 columns are assumed to be available and 5 names will be listed on each line. Smaller Width values will result in fewer names per line being displayed. A few examples follow:

```
+++FILES
+++FILES,1,A.T,FR
```

The first example would list all files on the working drive. The second example would list only those files on drive 1 whose names began with 'A' and extensions began with 'T', as well as those files whose names started with 'FR'.

FIND

The FIND command is used for finding all lines in a text file containing a specified string. It is faster to use FIND than to enter the editor to find strings.

DESCRIPTION

The general syntax of the FIND command is:

```
FIND,<file spec>,<string>
```

The file spec defaults to a TXT extension and to the working drive. The string may contain any printable (non-control) characters and is terminated by the carriage return or end of line character. Upon execution, all lines containing the specified string are printed on the terminal preceded by their line numbers. When finished, the total number of lines found containing the string is printed. Following are a few examples.

```
+++FIND,TEXT,THIS IS A TEST
+++FIND,BOOK.TXT,OHIO
```

The first example would find and display all lines in the file TEXT.TXT which contained the character string "THIS IS A TEST". The second example would search the file BOOK.TXT for the string "OHIO" and list all lines found.

FREE

The FREE command is used to report the total number of free (available) sectors on a diskette. The approximate number of kilobytes remaining is also reported.

DESCRIPTION

The general syntax of the FREE command is:

```
FREE[,<drive number>]
```

If the drive number is not specified it will default to the working drive. An example follows:

```
+++FREE,1
```

This command line will report the number of available sectors and approximate number of kilobytes remaining on the disk in drive 1.

MAP

The MAP utility is used for determining the load addresses and transfer address of a binary file. This command is useful in conjunction with the SAVE command.

DESCRIPTION

The general syntax of the MAP command is:

```
MAP,<file spec>
```

where the file spec defaults to a BIN extension and to the working drive. The beginning and ending addresses of each block of object code will be printed on the terminal. If a transfer address is contained in the file, it will be printed at the end of the list of addresses. If more than one transfer address is found in a file, only the effective one (the last one encountered) will be displayed. An example will demonstrate the use of MAP.

```
+++MAP,MONITOR
```

This command line would cause the load addresses and transfer address (if one exists) of the file named MONITOR.BIN to be displayed at the terminal.

The MEMEND command is used to interrogate or set the FLEX Memory End value. This value is used by many FLEX programs to determine the last memory location that they may use.

DESCRIPTION

The general syntax of the MEMEND command is:

```
MEMEND?              or
MEMEND               or
MEMEND <value>
```

The first form, that ending with a question mark, will cause the current value of Memory End to be printed at the terminal. The second form will cause MEMEND to perform a non-destructive test of memory to determine the highest usable address. It is assumed that the computer's memory is organized in 4K blocks starting at location zero. The Memory End value is set to that value which was determined to be the last address of contiguous memory by the memory test; it is also printed at the terminal. The third form, with a parameter, will set the Memory End value to that specified in the parameter. The parameter must be a hex value, without a leading dollar sign. Some examples follow:

```
+++MEMEND?
+++MEMEND
+++MEMEND 7FFF
```

The first example will cause the current value of Memory End to be printed at the terminal. The second example will cause a test to be performed, and Memory End to be set to the value determined by the test as the end of contiguous memory. The third example will cause the value of Memory End to be set to hex 7FFF.

The MEMEND utility should be used to reset Memory End whenever another program has modified it. An example is the abnormal exit from an EXEC procedure. EXEC changes Memory End to protect itself and, if aborted, will not restore it to its previous value.

PDEL

The PDEL command is a prompting delete utility. Either all files or only files matching a specified match list are displayed by name, one at a time, giving the option of deleting the file or keeping it. This command is very convenient for quickly removing a lot of no longer needed files from a disk.

DESCRIPTION

The general syntax of the PDEL command is:

```
PDEL[,<drive list>][,<match list>]
```

where drive list and match list are the same as described in the CAT command. Upon execution of PDEL, each file name will be printed at the terminal along with a delete request:

```
DELETE "FILE" ?
```

At this time three responses are valid. If a "N" is typed, the file will be left intact and the next name will be displayed. If a "Y" is typed, that file will be deleted. This utility DOES NOT ask if you are sure you want the file deleted, so make sure the first time! A carriage return may also be typed in response to the prompt at which time control will return back to FLEX. If a response other than one the three above is given, the delete request will be posted again. An example follows:

```
+++PDEL,1,.TXT
```

This command line would cause each file on drive 1 which has a TXT extension to be displayed and the delete option offered. Remember that once "Y" has been typed to the prompt, that file is gone forever!

RUN

The RUN command is used to load and optionally execute a position independent program at an address different from that at which the program normally executes.

DESCRIPTION

The general syntax of the RUN command is:

```
RUN,<load address>,<command>     or
RUN/<load address>,<command>
```

where <load address> is that location at which the command is to be executed, and <command> is the command, with associated parameters, that is to be executed. The second form indicated above (with the slash following RUN) will cause <command> to be loaded at <load address>, but not executed. In this form, control will return to FLEX after the program is loaded. Be aware that this program does not change any of the instructions in the program being loaded. If the program is truly position independent, it will execute correctly at the new load address. Some examples follow.

```
+++RUN,1000,DEBUG
+++RUN/3500,TEST
```

The first example will load the program DEBUG.CMD at address 1000 hex and start it executing. The second example will load the program TEST.CMD at address 3500 but will not execute it. Control will be returned to FLEX instead.

SPLIT

The SPLIT command is used to split a text file into two new files at a specified line number. It is convenient to use when a file becomes too large to easily manage or to break off an often-used section of text into another file.

DESCRIPTION

The general syntax of the SPLIT command is:

SPLIT,<input file spec>,<out file spec1>,<out file spec2>,<N>

The input file is the file to be split, output file spec 1 is the name to be assigned to the first set of lines read from the input file, output spec 2 is the name to be assigned to the rest of the file being split, and N is the line number at which the file should be split. The second output file will begin with line N of the input file. All files default to TXT extensions and to the working drive. An example follows:

+++SPLIT,TEST,TEST1,TEST2,125

This command line would cause lines 1 to 124 of the file named TEST.TXT on the working drive to be written into a file named TEST1.TXT and lines 125 to the end of the file to be written into a file named TEST2.TXT. The original file (TEST) remains unchanged.

ZAP

The ZAP command is a file delete utility. Either all files or only files matching a specified match list are deleted without any prompting. This command is very convenient for quickly removing a lot of no longer needed files from a disk.

DESCRIPTION

The general syntax of the ZAP command is:

```
ZAP[,<drive list>][,<match list>]
```

where drive list and match list are the same as described in the CAT command. Upon execution of ZAP, the name of each file deleted will be printed at the terminal in the form:

```
DELETING "FILE"
```

Be aware that there is no chance for "second thoughts". Once ZAP is invoked, the files will be deleted without any further intervention by the user. An example follows:

```
+++ZAP,1,.BAK
```

This command would cause all of the files on drive 1 with a .BAK extension to be deleted. It is wise, before invoking ZAP, to check which files will be deleted by doing a CAT, DIR, or FILES with the same match list that will be used with ZAP.